

# Handling Dynamic Schema Change in Process Models

Shazia W. Sadiq\*

Department of Computer Science and Electrical Engineering  
The University of Queensland  
QLD 4067 Australia  
email:shazia@csee.uq.edu.au

## Abstract

*Workflow technology has emerged as an appropriate platform for consolidating the distributed information resources of an enterprise, promoting interoperability across cross-platform systems and for providing a global view and understanding of business process models. However, the business processes that workflows represent, are dynamic by nature, that is, they encounter frequent and unavoidable changes. It is through this dynamism that organizations maintain their competitive edge. Workflow technology to date does not provide sufficient support for dynamically changing processes. Managing schema change of workflow processes with multiple active instances is a complex issue. In this paper, we present an analysis of workflow changes in relation to business process change, and present a classification of workflow changes that dictate the scope of the problem. Based on this classification we lay the foundation for a generic framework to support dynamically changing workflow processes.*

## 1. Introduction

Business environments have become exceedingly dynamic and competitive in recent times. Technological advancements have made it possible for organizations to automate their business processes extensively. Specialized information systems have emerged which are tailored to specific functional divisions of organizations. At the same time, there have been giant leaps in connectivity, owing to advancement in communication infrastructure and protocols. Organizations now seek to achieve higher goals through consolidating their specialized, but distributed, and often disconnected, resources. Workflow technology provides an appropriate platform for consolidating the distributed information resources of an enterprise and providing a global view and understanding of business process models.

A Business Process model is a description of an organization's activities in terms of tasks, agents, rules

and procedures. It is engineered to fulfill a business goal [1]. A workflow is defined as the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules [2]. Workflows thus bring together the tasks, agents, rules and procedures of a business process. Workflow management is a means by which the ordering, coordination and allocation of tasks can be defined and controlled in accordance with usually a given set of rules and procedures. The agents, also called processing entities, responsible for performing these tasks can be humans and/or information systems. A Workflow has two main components:

*Process Model* - The process model, or workflow model is a definition of the tasks, ordering, data, resources, and other aspects of the process. This represents the workflow **schema** or type. Most, if not all, workflow models are defined as graphs which depict the flow or ordering of the tasks involved in the process, together with a description of other task properties. For example, we can define an admission workflow that handles student admission applications in a university.

*Process Instance* - The process instance is a particular **occurrence** of the process, for example, a particular application for admission represents an instance of the admission workflow. Different instances of the same workflow may perform a different subset of workflow tasks, i.e. they may have different execution paths in the workflow graph. An *instance class* is a set of instances that can be represented by the same (execution) sub-graph.

Although exception handling has been an active area of research in information systems [3], [4] recently, the issue has attracted interest in workflow research groups, owing mostly perhaps to its necessity [5], [6], [7]. One of the earliest contributions came from [8]. They divide exceptions that may occur during workflow execution

---

\* The work reported in this paper has been funded in part by the Cooperative Research Centres Program through the Department of the Prime Minister and Cabinet of the Commonwealth Government of Australia.

into basic failures, which are failures at the system level, application failures, expected exceptions and unexpected exceptions. Recovery from *system* failure such as power cut, program abort, server down etc. is generally handled by the workflow activity which relies on the recovery capabilities of underlying (database management) systems. The workflow may be temporarily suspended, but resumes execution when the local system recovers. We do not discuss this aspect of Workflow Management System (WFMS). *Semantic* failure occurs when an instance is unable to proceed according to the given workflow model. This may happen at the application (task) or process (workflow) level. Thus the workflow system is unable to cater for the special circumstances. Workflow changes may be initiated as a result of semantic failure. Such failures may be identified by workflow clients when an exceptional instance is encountered, and/or by external entities when the process (model) changes, affecting all or some instances.

The problem of dynamic schema change in workflows is two-fold: Firstly, it has to be determined what changes are to be made to the business process so as to capture the new or exceptional situation. Secondly, these changes are to be propagated down to the underlying workflow implementations. This paper deals with the latter. In the following sections we first introduce a taxonomy of workflow changes. This taxonomy dictates the scope of the problem. We will then present a modification methodology that aims to cater for the different types of workflow changes under essentially the same framework.

## 2. Effects of Business Process Change on Workflows

An important distinction in business changes affecting underlying workflows systems is whether modifications are to be made to the workflow model, particular instances, or both. Changes made to the workflow model indicate a permanent change of the business process as a result of process improvement [9], process innovation or business process reengineering, or simply because of design errors. Changes that affect only particular or few instances represent unforeseen, presumably rare situations in the business process.

As a first step towards understanding the scope of workflow modifications, we identify five related but distinct types of workflow changes. We view these changes as *Modification Policies*, which can be adopted by the workflow administrator (WFA). When business processes change, because of some event internal or external to the organization, the changes are generally planned, revised, approved and specified by high level managers or consultants, and then propagated to operational level. We see the role of the WFA as a mediator between management's proposals and strategies, and the propagation of these proposals to the operational level. WFA thus has to be capable of translating process changes into workflow models, and making decisions

regarding handling of active workflow instances. Modification Policies, which may be adopted by the WFA, are:

*Flush*: In flush situations all current instances are allowed to complete according to the old process model, but new instances are planned to follow new model. New instances may be put on hold, until all current instances have completed. However, the two specifications could also be allowed exist simultaneously.

*Abort*: Active workflow instances may be aborted when the process model is changed. Abort is most commonly used for adaptation of individual instances, for example canceling a reservation. Abort may incur losses to the organization, and in some cases the losses may be unacceptable. In most cases, abort will require some compensation for the work accomplished so far.

*Migrate*: A change that effects all current instances, may have to be introduced without allowing the instances to abort or flush. Current instances would normally be in different stages of process execution.

The problem arises when an instance is at a stage where tasks already accomplished have affected the process in such a way that subsequent tasks are unable to proceed in accordance with the new specification. Thus migration may involve undo or compensation of completed tasks, in order to bring the instance in compliance with the new specification. The worst case is when the complete process has to be rolled back to the start, that is all work is lost or undone. This special case is equivalent to Abort.

*Adapt*: Adapt includes cases of errors and exceptions, where the process model may not change permanently, but some instances have to be treated differently because of some exceptional and unforeseen circumstances.

*Build*: Building of a new process is also a class of process change. The difference is that the starting point is not a detailed pre-existing model, but an elementary description. The advantage of including build as a class of process change, is that it allows the inclusion of processes which cannot be fully predefined, into the domain of process change.

The common denominator in all of the above policies, with the possible exception of flush, is that they effect active instances of the given process model. Thus they dictate the scope of workflow modification and constitute *dynamic* modification, in contrast to *static* modification, which is merely a change in the workflow model, i.e. no currently active instances are involved.

### 3. Modification Methodology

We propose a simple yet effective methodology based on a three-phase modification process that consists of defining, conforming to and enacting the modification. We will briefly describe the three phases in the following sections. The main motivating factors for this methodology are as follows:

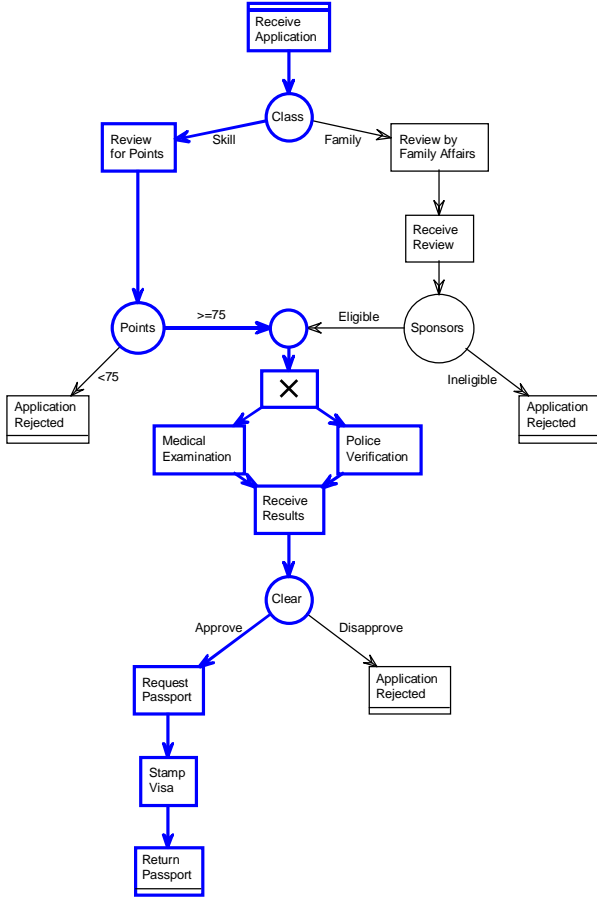


Figure 1. Example Immigration

- To provide support for the modification policies so that changes to individual instances (adapt), schema evolution (migrate), flush, abort and build, all can be tackled under essentially the same framework.
- To explore and determine the extent of automation that can be provided for the propagation of business process change to underlying workflow implementations, thus minimizing need for user involvement and ultimately minimizing costs.

We briefly introduce the workflow modeling language first [10]. This conforms closely to the Workflow Management Coalition standards [11]. Figure 1 gives an example for an immigration workflow that handles immigration visa applications.

A Workflow is a Directed Acyclic Graph (DAG)  $\mathbf{W} = \langle \mathbf{N}, \mathbf{F} \rangle$  such that  $\mathbf{N}$ : Finite Set of Nodes,  $\mathbf{F}$ : Flow Relation  $\mathbf{F} \subseteq \mathbf{N} \times \mathbf{N}$ ,

Nat: Set of Natural Numbers,

$\forall n \in \mathbf{N}, I: \mathbf{N} \rightarrow \text{Nat}, I(n)$  : Number of incoming flows for node  $n$

$\forall n \in \mathbf{N}, O: \mathbf{N} \rightarrow \text{Nat}, O(n)$  : Number of outgoing flows for node  $n$

$\mathbf{N} = \mathbf{C} \cup \mathbf{T}, \mathbf{C} \cap \mathbf{T} = \phi$  where  $\mathbf{C}$ : Set of Condition Nodes,  $\mathbf{T}$ : Set of Task Nodes

$\forall n \in \mathbf{T}, \text{s.t. } I(n)=0 \wedge (\neg \exists m \in \mathbf{T}, \text{s.t. } I(m)=0 \wedge m \neq n)$ , we call this Initial Node  $n_0$

$\forall n \in \mathbf{T}, \text{s.t. } O(n)=0 \wedge (\neg \exists m \in \mathbf{T}, \text{s.t. } O(m)=0 \wedge m \neq n)$ , we call this Final Node  $n_f$

$\forall n \in \mathbf{N}, \exists \mathbf{P}, \text{s.t. } \mathbf{P} = \{n_0, \dots, n_f\}$

$\forall n \in \mathbf{C}, I(n) \geq 2 \vee O(n) \geq 2$

Each task in the workflow is described by a set of properties. These properties relate to the data, time, underlying applications, resources, clients, compensation and much more [12]. We do not elaborate on task properties in this paper. However, the task is a complex object with rich semantics, and cannot be considered as a mere node in the workflow graph.

Each node  $n \in \mathbf{N}$  in the Workflow  $\mathbf{W}$  also has an execution structure consisting of a set of visible states and a set of transitions between these states [13]. [14]. The execution of nodes is modeled by the Finite State Machine in Figure 2. Nodes can take the following states:

*Scheduled*: The activity is scheduled for execution, but has not been allocated to the workflow client.

*Active*: The activity designated to the node has been chosen by its client, and is being performed.

*Suspend*: Temporarily put on hold.

*Complete*: Successful completion.

*Terminated*: The result of aborting an active task or recalling a scheduled task.

Let  $\mathbf{INS}$  be the set of instances for  $\mathbf{W}$

nodestate:  $\mathbf{N} \times \mathbf{INS} \rightarrow \{\text{Scheduled, Active, Complete, Terminated, Suspended}\}$ ,

nodestate( $n, i$ ) : State of Node  $n$  for instance  $i$  where  $n \in \mathbf{N}, i \in \mathbf{INS}$ .

The execution of  $\mathbf{W}$  is driven by the following rules. The aim here is to model the scheduling and progress of the workflow, and not the execution of its underlying activities. These rules capture the semantics of the various constructs used in the modeling language. We do not elaborate upon the properties of these constructs in this paper, other than to give a quick understanding to the user with regards to workflow execution.

**Initialization Rule:**

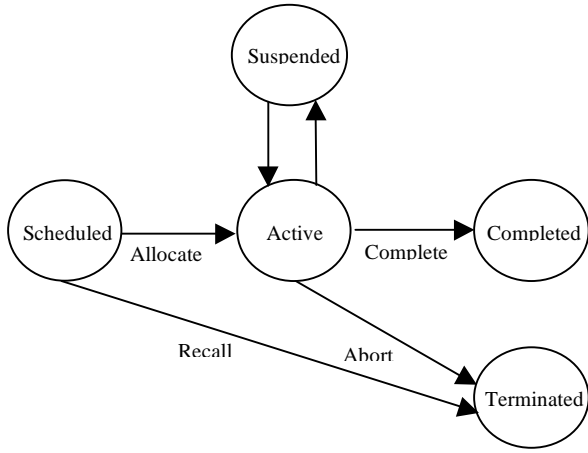
An instance  $i$  is initialized when  $nodestate(n_0, i) = \text{Scheduled}$ .

**Scheduling Rule:**

Task nodes of an instance  $i$  are scheduled when all incoming flows have been triggered (And-Join). Condition nodes of an instance  $i$  are scheduled when one incoming flow has been triggered (Or Join)

**Completion Rule:**

All outgoing flows of a Task node of an instance  $i$  are triggered upon it reaching the Complete state (And Split, Sequential execution). One outgoing flows of a Condition node of an instance  $i$  is triggered upon it reaching the Complete state (Or Split)



**Figure 2. Finite State Machine for a Node**

Assuming that time lapse between the completion of a node and the scheduling of its ‘chosen’ successors is negligible, we define the following states for an instance:

**Initial:** An instance  $i$  is in Initial State when  $nodestate(n_0, i) = \text{Schedule}$

**Current:** An instance  $i$  is in Current State when  $\exists n \in \mathbf{N}$ , s.t.  $nodestate(n, i) = \text{Active} \vee (\exists n \in \mathbf{N}$ , s.t.  $nodestate(n, i) = \text{Scheduled} \wedge \exists m \in \mathbf{N}$ , s.t.  $nodestate(m, i) = \text{Complete}$ )

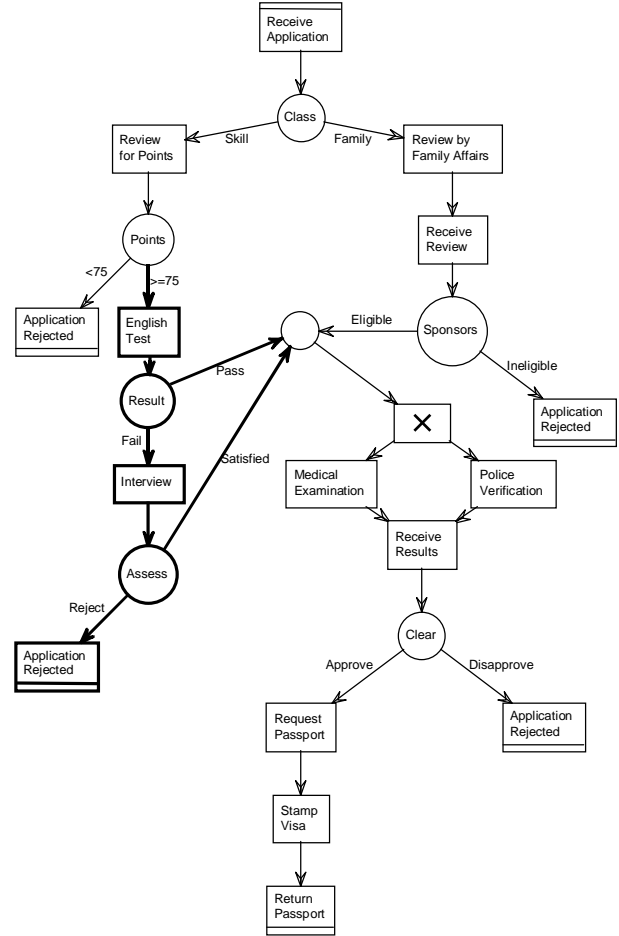
**Complete:** An instance  $i$  is in Complete State when  $nodestate(n_i, i) = \text{Complete}$

**3.1. Defining the Modification**

The modification process begins by defining the modification, which constitutes specifying the

*modification policy*, specifying the *affected process instances* in case of abort or adapt, and specifying the *changes to the process model*.

Changes to the process model can comprise of inserting new tasks, removing old tasks, changing the task properties (data requirements, underlying application, temporal constraints, resource allocation etc.), or modifying the order of task execution. We define  $\mathbf{M}$  to be the Modification on the Workflow  $\mathbf{W}$ , such that  $\mathbf{M}: \mathbf{W}^k \rightarrow \mathbf{W}^{k+1}$  where  $\mathbf{W}^k$  is Workflow Model Version  $k$ .  $\mathbf{M}$  consists of a sequence of *prescribed operations* which when performed upon  $\mathbf{W}^k$ , will give  $\mathbf{W}^{k+1}$ .



**Figure 3. Modified Immigration Workflow**

There are two obvious constraints in this regard: the changes to be made must be known, and the changes must be verified in accordance with the correctness properties of the modeling language being used. Significant research has been done on the conceptual specification of workflow model and provision of change operations which guarantee the (structural) correctness of the changed model [15], [16], [17], [18]. Our contention is that modifying the workflow (model) should be as flexible as building a new one, as long as the changes are

verified before the instances are involved. Since no instances are involved during the modification of the model, condition for correctness after each operation may unnecessarily restrict the modifier. Our approach relies on a verification tool which works with the process definition tool. This allows the user to flexibly define the requisite changes, and verify them after any interval. System verification is imposed at commit. The correctness properties and verification algorithms for the above language can be found in [19].

These have been implemented in a prototype FlowMake, which is a graphical modeling tool supported by a verification engine that verifies the workflow graph by identifying structural conflicts.

### 3.2. Conforming to the Modification

After defining the modification, the next step is to bring the affected instances of  $\mathbf{W}^k$  in conformity with the specifications of  $\mathbf{W}^{k+1}$ . To begin with, instances must be grouped, otherwise the process will reduce to individual handling of every affected instance. We proposed a three level *grouping scheme*. At the first level, instances are grouped with respect to their *class*. At the second level instances are grouped on the basis of *compliance*. Compliance basically means that either the instance has not yet reached the stage of the workflow which is affected by the modification or because of a condition outcome, the instance has taken a path which is not affected by the modification. In other words, the entire execution trace of the instance can be found in  $\mathbf{W}^{k+1}$ . At the third level the *stage* of the non-compliant instances is determined. It would be unrealistic to assume that there would never be a situation during the modification of active instances where external intervention will not be required. It is conceivable that instances may be at such a stage that trying to adjust them will be too difficult, or equivalent to abort. These instances may have to be handled externally. At the third level, instances are grouped on this basis, i.e. whether compliance can be achieved within the system, or externally.

We further propose the concept of *Compliance Graphs* for affected instances. The Compliance Graph for an instance initialized under  $\mathbf{W}^k$ , defines a bridge between  $\mathbf{W}^k$  and  $\mathbf{W}^{k+1}$ . The instance follows a unique path which consists partially of  $\mathbf{W}^k$ , the compliance graph, and partially of  $\mathbf{W}^{k+1}$ . A pre-requisite for the definition of compliance graphs, is the specification of transactional properties for workflow tasks. Although the definition of task properties is beyond the scope of this paper, we identify the following simple properties in order to illustrate the modification process.

$\forall n \in \mathbf{N}$  Compensable:  $\mathbf{N} \rightarrow \{Y, N\}$  where Compensable (n) = Y indicates that the activity represented by this node can be compensated. If Compensable (n) = Y, then Compensation-Task (n) is the compensation task for n.

Suppose an instance  $i$  initiated in  $\mathbf{W}^k$  (Fig 1) is currently executing the task 'Request Passport'. The

immigration process is changed, resulting in a new workflow model  $\mathbf{W}^{k+1}$  (Figure 3). All active instances including  $i$  are to be *migrated* to the new process. Figure 4 demonstrates the concept of compliance graph for the instance  $i$ . The task 'Return Passport with letter of Explanation' is a compensation action for 'Request Passport'. Other affected tasks are 'Medical Examination' and 'Police Verification. However, these are not undone (or redone) because of transactional properties specified for these tasks, for example, how can we undo medical examination?

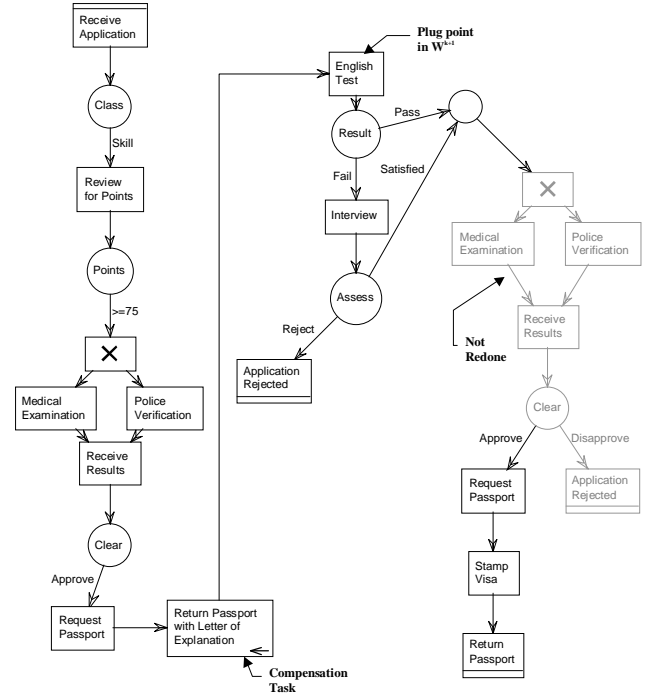


Figure 4. Using Compliance Graphs

Compliance graphs thus provide revised schedules, which chart out the plan until completion for affected instances. Thus, except for the (hopefully small) group of instances identified in the level 3 grouping, the workflow management system can continue to provide automated support for the changed business process. This phase is no doubt the most critical and challenging. The strength of the presented methodology lies in its ability to uniformly support all classes of workflow modification.

Let the stage of an instance  $i$  of  $\mathbf{W}^k$  be represented by the set  $\mathbf{N}^i$  s.t.  $\forall n \in \mathbf{N}^i$ , nodestate (n, i) = Active or Completed

Let  $\mathbf{N}^{\text{Current}} \subseteq \mathbf{N}$  of  $\mathbf{W}^k$  be the set of nodes in an instance  $i$  s.t.  $\forall n \in \mathbf{N}^{\text{Current}}$ , nodestate (n, i) = Active or Scheduled

The construction of the compliance graph begins with handling the nodes of  $\mathbf{N}^{\text{Current}}$ . Generally, all scheduled nodes will be recalled i.e. removed from work lists, and all active nodes will be aborted by external intervention. In some cases active nodes may be allowed to continue if

they exist in  $W^{k+1}$  and ordering dependency is not violated. However, this can be considered as a performance enhancement. We give CG-DEFINE<sup>1</sup>, the procedure for determining:

$N^{Comp}$ : Set of compensation activities required for rollback

$N^{Plug}$ : Plug nodes in the new model

Let  $N^{Comp} = N^i$  and  $N^{Plug} = \{\}$ . STACK is a stack of nodes. StackIn (n) is a procedure to insert node n in STACK and StackOut (n) is a procedure to remove node n from STACK. GetStack is a procedure that gets the last-in element from STACK. Let Succ(n) be the set of successors of n.

```

StackIn (n0)
CG-DEFINE
  Let n = GetStack
  IF Incoming and Outgoing flows of n are same in
Wk and Wk+1
  THEN  NComp = NComp - n
        StackOut (n)
        ∀ m ∈ Succ(n) in Wk, IF m ∈ NComp
THEN StackIn (m)
  ELSE  NComp = NComp - n
        NPlug = NPlug + n
        StackOut (n)
  ENDIF
  IF STACK is empty
  THEN  Exit
  ELSE  GO TO CG-DEFINE
  ENDIF
END CG_DEFINE

```

As a result of this procedure, the two sets  $N^{Comp}$  and  $N^{Plug}$  are determined. In the example given in Fig 4.,

$N^{Current} = \{\text{Request Passport}\}$

$N^i = \{\text{Receive Application, Class?, Review for Points, Points?, Medical Examination, Police Verification, Receive Results, Clear?, Request Passport}\}$

CG-DEFINE determines the following:

$N^{Comp} = \{\text{Medical Examination, Police Verification, Receive Results, Clear?, Request Passport}\}$

$N^{Plug} = \{\text{English Test}\}$

We then remove the nodes  $n \in N^{Comp}$  for which Compensable (n) = N. And we substitute the remaining nodes with Compensation-Task (n). This reduces  $N^{Comp}$  to

$N^{Comp} = \{\text{Return Passport with Letter of Explanation}\}$ .

By following the reverse partial order of execution of the nodes in  $N^{Comp}$ , we generate a graph, that links  $N^{Current}$  to  $N^{Plug}$  through  $N^{Comp}$ .

### 3.3. Enacting the Modification

The last phase in the modification procedure is that of enacting the modification. This relates to the handling of workflow execution during the transition period. The transition period is signified by instances that started with  $W^k$  but are still executing. Instances may follow  $W^k$  (e.g. in flush), or  $W^{k+1}$  (for instances initialized after modification has been defined), or revised schedules based on compliance graphs. Instance execution, which may have been put on hold, will restart. When all instances following old model or revised schedules have completed, the new model becomes the current workflow model, and the modification process is complete.

## 4. Contribution

The goal of this paper was to present foundation concepts for dynamic schema change of workflows, and provide feasibility considerations for the automation of this process. To achieve this, we identified various classes of change for workflows. These are Flush, Abort, Migrate, Abort and Build. We further, present a modification methodology that is capable of handling these classes of change under essentially the same framework. The methodology follows a three-phase procedure of defining, conforming to and enacting the modification.

The second phase of Conforming is no doubt the most important and challenging. We introduced the concept of Compliance Graphs and showed how these graphs can be constructed and used for supporting dynamic modification.

## References

- [1] Georgakopoulos D., Hornick M., Sheth A. (1995) An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. Journal on Distributed and Parallel Databases, 3(2):119-153.
- [2] Workflow Management Coalition (1995) The Workflow Reference Model, Document Number TC00-1003, Issue 1.1, 19-Jan-95.
- [3] Saastomoinen Heikki, White George M. (1995) On Handling Exceptions, Proceedings of ACM Conference on Organizational Computing Systems (COOCS 95), Milpitas, CA. USA, Nov 1995.
- [4] Strong Diane M., Miller Steven M. (1995) Exceptions and Exception Handling in Computerized Information Processes, ACM Transactions on Information Systems, Vol. 13, No 2, Pages 206-233, April 1995.
- [5] Casati Fabio., Fugini Mariagrazia, Mirbel Isabelle (1999) An environment for designing exceptions in workflows. Information Systems 24(3) Pages 255-273, 1999.

<sup>1</sup> Details regarding synchronization of multiple current nodes and/or multiple plug nodes have been skipped for simplicity

- [6] Faustmann Gert (1998) Enforcement vs. Freedom of Action – An Integrated Approach to Flexible Workflow Enactment. Towards Adaptive Workflow Systems, workshop at the Conference on Computer Supported Cooperative Work. Seattle, WA, USA. November 1998.
- [7] Han Yanbo, Sheth Amit (1998) A Taxonomy on Adaptive Workflow Management. Towards Adaptive Workflow Systems, workshop at the Conference on Computer Supported Cooperative Work. Seattle, WA, USA. November 1998.
- [8] Eder J., Liebhart W. (1995) The workflow activity model WAMO. Proceedings of the 3rd international conference on Cooperative Information Systems (CoopIS), Vienna, Austria, May 1995.
- [9] Davenport Thomas (1993) Process Innovation – Reengineering work through Information Technology. Harvard Business School Press.
- [10] Sadiq Wasim, Orłowska Maria E. (1997) On Correctness Issues in Conceptual Modeling of Workflows. In Proceedings of the 5<sup>th</sup> European Conference on Information Systems (ECIS '97), Cork, Ireland, June 19-21, 1997.
- [11] Workflow Management Coalition (1998) Interface 1: Process Definition Interchange, Process Model, Document Number WfMC TC-1016-p.
- [12] Jablonski Stefan, Bussler Christopher (1996) Workflow Management – Modeling Concepts, Architecture and Implementation. International Thompson Computer Press 1996.
- [13] Rusinkiewicz M., Sheth A. (1994) Specification and Execution of Transactional Workflows. In W. Kim, editor, Modern Database Systems: The Object Model, Interoperability, and Beyond. Addison Wesley.
- [14] Kuo Dean, Lawley Michael, Liu Chengfei, Orłowska Maria E. (1996) A General Model for Nested Transactional Workflow. Proceedings of the International Workshop on Advanced Transaction Models and Architecture (ATMA'96), Bombay India, pp.18-35, 1996.
- [15] Aalst W.M.P. van der (1997) Verification of Workflow Nets. In Application and Theory of Petri Nets, Editors P. Azema and G. Balbo, Volume 1248 of Lecture Notes in Computer Science.
- [16] Casati F., Ceri S., Pernici B., Pozzi G. (1996) Workflow Evolution. In Proceedings of the 15<sup>th</sup> International Conference on Conceptual Modeling, ER'96, Cottbus, Germany. Springer Verlag, Lecture Notes in Computer Science.
- [17] Ellis S., Keddara K., Rozenberg G. (1995) Dynamic Changes within Workflow Systems. Proceedings of ACM Conference on Organizational Computing Systems (COOCS 95).
- [18] Reichert Manfred, Dadam Peter (1997) ADEPTflex - Supporting Dynamic Changes of Workflow without losing control. Journal of Intelligent Information Systems (JIIS), Special Issue on Workflow and Process Management.
- [19] Sadiq Wasim, Orłowska Maria E. (1999) Applying Graph Reduction Techniques for Identifying Structural Conflicts in Process Models, 11th Conference on Advanced Information Systems Engineering (CAiSE99). Heidelberg, Germany. June 1999.